**Programming project documentation:**

By *Hamzah Saddiq*

Contents:

[Initial analysis (unfinished draft): 4](#_Toc65534778)

[A computational problem (unfinished draft): 4](#_Toc65534779)

[Amenability of my project as a computational problem: 5](#_Toc65534780)

[Stakeholders (need to make up results): 6](#_Toc65534781)

[Stakeholder identification: 6](#_Toc65534782)

[Stakeholder interview (need to make up results): 7](#_Toc65534783)

[Results: 8](#_Toc65534784)

[Stakeholder further contact: 10](#_Toc65534785)

[Problem research (finished draft): 11](#_Toc65534786)

[Previous solutions: 11](#_Toc65534787)

[Solution essential features: 11](#_Toc65534788)

[Solution limitations: 12](#_Toc65534789)

[Proposed solution (unfinished draft): 12](#_Toc65534790)

[Solution requirements: 12](#_Toc65534791)

[Final solution: 12](#_Toc65534792)

[Solution success criteria: 14](#_Toc65534793)

[Initial design (unfinished draft). 15](#_Toc65534794)

[Problem decomposition (unfinished draft): 15](#_Toc65534795)

[Solution description (unfinished draft): 17](#_Toc65534796)

[Solution structure. 17](#_Toc65534797)

[Inputs, processes, outputs and storage. 18](#_Toc65534798)

[Algorithms: 19](#_Toc65534799)

[Sign up module: 19](#_Toc65534800)

[Sign in module: 21](#_Toc65534801)

[Leader board module: 22](#_Toc65534802)

[Usability features: 24](#_Toc65534803)

[Main menu: 24](#_Toc65534804)

[Main settings menu: 24](#_Toc65534805)

[Accessibility settings menu: 25](#_Toc65534806)

[Key variables and structures: 25](#_Toc65534807)

[Classes: 27](#_Toc65534808)

[Testing approach. 28](#_Toc65534809)

[Development: 31](#_Toc65534810)

[First prototype: 31](#_Toc65534811)

[Analysis. 31](#_Toc65534812)

[Design. 31](#_Toc65534813)

[Test data. 31](#_Toc65534814)

[Implementation. 32](#_Toc65534815)

[Quitting procedure. 32](#_Toc65534816)

[Background drawing: 33](#_Toc65534817)

[Rendering text: 33](#_Toc65534818)

[Testing: 34](#_Toc65534819)

[Remedial actions. 34](#_Toc65534820)

[Evaluation. 34](#_Toc65534821)

[Stakeholder contact: 35](#_Toc65534822)

[Second prototype: 35](#_Toc65534823)

[Analysis. 35](#_Toc65534824)

[Design. 36](#_Toc65534825)

[Test data. 36](#_Toc65534826)

[Implementation. 37](#_Toc65534827)

[Page layout: 37](#_Toc65534828)

[Hover method: 37](#_Toc65534829)

[Image button implementation: 38](#_Toc65534830)

[Testing. (empty) 38](#_Toc65534831)

[Remedial actions. (empty) 38](#_Toc65534832)

[Evaluation. (empty) 39](#_Toc65534833)

[Stakeholder contact. (empty) 39](#_Toc65534834)

[Third prototype. 39](#_Toc65534835)

[Analysis (empty). 39](#_Toc65534836)

[Design (empty). 39](#_Toc65534837)

[Test data (empty). 39](#_Toc65534838)

[Implementation (empty). 39](#_Toc65534839)

[Testing (empty). 39](#_Toc65534840)

[Remedial actions (empty). 40](#_Toc65534841)

[Evaluation (empty). 40](#_Toc65534842)

[Stakeholder contact (empty). 40](#_Toc65534843)

[References 41](#_Toc65534844)

# Initial analysis:

## A computational problem:

My project is essentially a high score-based, sideways platformer. The problem my project is based on is computational. Here are a few reasons why.

A computational project should involve calculations. For example, with my project the player’s position, the conditions randomly generated platforms must meet, and projectile positions, are all based on various, multiple-layered calculations.

A computational problem should involve inputs, outputs and processes. There are several inputs required at multiple stages of my project- including the start menu; the options menu; the main game; leader board consent; leader board details, account details, gameplay inputs and leader board access. Similarly, there are several outputs required at multiple stages too, including displaying the user interfaces for: the start menu, options menu, the main game, and the leader board. Other outputs such as sound effects would also be important to help the program function. It also involves logical reasoning to process said inputs into those outputs. ­This includes the reasoning that decides whether processes such as player death and movement, also deciding when they happen and even to what extent in certain situations.

A computational project should involve logical reasoning. Most aspects, internal and external processes that make up the project requi­­­­re extensive Boolean logic. Examples of this for my project include:

* checking criteria are met for eligible profile-related inputs regarding recording high scores
* checking the player has not collided with any obstacles
* calculating the position of user and enemy projectiles

A computational project should have a finite algorithm. My project has a set algorithm which is indefinitely terminated under the condition that the user exits the game window.

A computational project should involve concise instructions based on a formal language. My project algorithm is also concise to convert to a formal programming language. The pseudo-code involves many commands that resembles those from languages such as Python. Also, my project’s logic and reasoning are almost exclusively mathematical. This is because many processes of the game such as spawning platforms, enemies, projectiles etc. Are heavily maths-based which is also easy to convert to concise instructions based on a formal language. there is not any human reasoning or subjectivity in my algorithm. Therefore, this project could be efficiently converted from its algorithm form to a mixture of procedural and object-oriented languages in a straightforward manner.

A computational project should return the same output for a given input, every time: This project would have set algorithms that produce the exact same output for the same input, this is definite as the output would be based on calculations done involving the input. For example, pressing the jump keyboard assignment will always result in a jump attempt from the character.

A computational project should work regardless of the input: This project would be robust using loops, errors and exceptions to fully ensure that no possible inputs will cause any runtime or logic errors. Validation, such as the password validation for the account section, is also used to make the program more robust.

A computational project should halt at some point and produce a correct output: This program would end when the user inputs a close request, which would be done by clicking a close button on a GUI the form of a button on a window

A computational problem can be solved via the 3 computational processes: abstraction, decomposition and divide and conquer. Abstraction is the process of removing unnecessary details from software. The concept of the project itself has already been abstracted from real-life. Examples of this include, taking a 3D environment and abstracting it to 2D, restricting character movement to left to right, reducing enemy intelligence etc. The project could also be even further abstracted in numerous ways. For example, the sprites for the character, enemy, platforms and background have most of the detail removed. Decomposition is the process of splitting up a problem into smaller sub-problems which in turn may be further split, making difficult problems easier to solve. This problem is straightforward to break down into smaller sub-problems. For example, it could be broken down into the menu, playtime and high score sub-problems. High score could be split up into user consent, database overwriting and database display sub-problems. Therefore, my project is very simple to decompose. Dividing and conquering is the process of dividing a problem into smaller, independent modules which can be approached, tested and tweaked independently of each other. This project would be simple and direct to divide and conquer with a variety of potential, independent and programmable, modules.

### Amenability of my project as a computational problem:

­

My project is more efficiently solved computationally than without the involvement of computers.

One reason why is because It involves calculations. This problem involves many simple calculations that need to be completed consistently, flawlessly and rapidly so that values can be calculated fast enough in real time to make split second decisions based on the user’s i­nputs. Examples of these calculations include the user position, enemy positions, projectile positions, enemy spawn position and platform position. This is more efficiently done by a computer as a CPU can do billions of these every second therefore it’s needed to keep up with the demand of the algorithm. Also, humans are prone to errors and mistakes whereas a computer is not. Such errors would break the algorithm and are therefore unacceptable, therefore a computer would inarguably be more efficient at solving this problem and so my project is amenable as a computational problem.

Another reason is that it involves inputs, outputs and processes. Examples of these inputs include account details, gameplay inputs and menu navigation inputs. Examples of outputs include, displaying images on a monitor and producing sound effects through a speaker system. A CPU is capable processing these billions of times per second tirelessly, something which would require massive teams of people to adequately produce those results, even then mistakes would likely be made due to human error. Also, as all the processing is logic-based and requires zero human reasoning, there are no drawbacks of computationally processing the various components of my project. Therefore, a computer would be more efficient, and this further supports my project is amenable as a computational problem.

Another reason is that it Involves concise instructions based on a formal language. This removes the drawback, that many developers face when trying to develop software to solve a problem as no workarounds are needed when trying to implement the algorithm. Therefore, my project is a lot of the inefficiency regarding computationally solving a problem is removed and so my project is amenable as a computational problem.

Another reason is that it returns the same output for a given input, every time. My project is comprised of consistent processes and logic. For example, the character will always take damage when hit by a projectile and always attempt to jump when the jump button is pressed. Therefore, it is also consistent in the output it returns for a given input, making my project amenable as a computational problem.

Another reason is that it works regardless of the input. My project algorithm is robust and has several fail-safes such as error and exceptions or even loops to force input corrections rather than breaking down. An example of these being used is when validating text inputs for the account section. This negates most of the difficulty’s computer programs present when accommodating for user errors, further proving that my project is amenable as a computational problem.

Lastly, it Involves the 3 computational processes. My project algorithm is clear and concise to implement computational processes. For example, abstraction is used when simplifying the character models to sprites. Decomposition is used when breaking down the account problem into signing in, validation and signing out. Dividing and conquering is done to separately code, and hence the problems of the settings and account modules. This makes the process of programming it straightforward and faster. This makes it more efficient and amenable as a computational problem.

## Stakeholders:

### Stakeholder identification:

The stakeholders would consist of casual gamers. My project is aimed at those aged 10-40 during free time at home, break hours etc.

It is natural for an average person to quickly judge a seemingly bland game as poor and refuse to play it. Therefore, my project must be visually stimulating in order to entice the audience and stop them from rejecting it at first glance. For the same reason, my game also must be easy to get a hang of in order to reduce that same threat of deterring potential players early on.

The target audience would likely play for 5-30-minute sessions as they are convenient, do not require much commitment, and are appealing to those wanting quick fun. Therefore, the game needs to stimulate people in a way that would quickly gratify them and provide an easy way to blow off some steam.

The game would have to have many features tailored toward providing the target audience with a hassle-free, appropriate gaming experience.

One way of doing this is to make it compatible with desktop computers. Most have plentiful access to desktop computers at the places they spend most of their time (home, school, work etc.). Therefore, a platform to run the program will be provided without the need for anything they do not already have access to. However, it would still need fast loading times, as this makes it a lot more convenient to use considering users may only get several minutes at a time to play, so they cannot afford to spend a significant portion of that, waiting for the game to load. For this to be possible, small file size, processing requirements are required as to not demand to much from budget devices not designed for gaming.

A simple objective is also needed. The objective cannot be too complex as this would clash with the short session periods, trying to cram a complex objective within a short time frame would reduce game quality. Very simple and intuitive game controls and objectives would also be needed to work with the short session periods. This will contribute towards quickly engaging younger audiences, and those who are not able to spend long amounts of time and effort to get the enjoyment this game is made to provide.

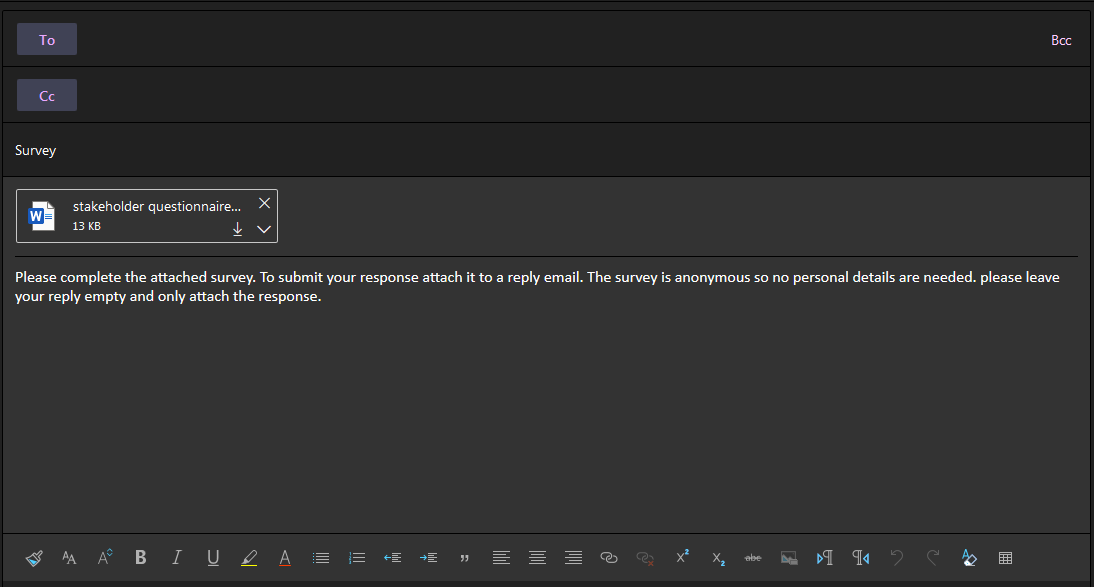
As the project is made for a wide age range graphics should be simple with arcade-like physics that still uniquely test timing and skill, as previously mentioned. Therefore, the game can be easy to learn and ‘get hang of’ as to avoid deterring users at early stage but still require skill to be acquired via playing time as to provide an objective to aim towards, which would be beating leader boards and high scores (hence the high score design). Also, the soundtrack composed of free synth wave instrumental music under creative commons licenses as that appeals to a wide age range and can be looped without feeling drawn out and repetitive.

It is also designed to provide an alternative to potentially damaging games that may increase procrastination, waste time etc. This is important as alternative games are helpful for reducing negative social stigmas around games (e.g., time-wasting, cut off social life, are not ever played in moderation), this suits the project’s short session design.

### Stakeholder interview:

Questionnaires will be sent to several individuals that fit stakeholder criteria. This would be in the form emails sent to those who fit the criteria for my audience (mainly those who play games between 10 to 40 years old) who in turn could get responses from them. (will make up a ‘stakeholder sample’). This questionnaire Will be sent and responded to via email. I will then use the responses to help design my game to be tailored to my overall stakeholder population. The initial survey will consist of five questions:

* **Q1:** *“How long would you say your individual break periods at work and home are?”* – This will help me tailor the game to be played for a certain length of time. For example, this could be done by adapting enemy variation: longer play lengths will require more variation within the gameplay to keep user engaged; whilst shorter lengths will mean that too much variation may prevent user from being able to learn to play it.
* **Q2:** *“What kind of variation would best stop you from getting bored of a game of this type?”* – This will prevent me from adding unnecessary complexity that would confuse my end consumers and also not including enough to keep the child engaged and cognitively stimulated (this would also hinder the player’s ability to switch from playing back to work, errands, housework etc. as the brain will have been too inactive)



#### Results:

* For this questionnaire I got 30 total responses.
* Here is a pie chart explaining the results gathered:
* Each minute range represents a group :
  + 0-5 : This group got 2 responses, only accounting for around 7% of my stakeholder sample. Therefore, I can conclude that only a small portion of my end consumers will be running the code for less than 5 minutes.
  + 5-10 : This group got 10 responses, accounting for 33.% (a third) of my stakeholder sample. Therefore, I can conclude that a significant portion of my end consumers will be running the code for 5 to 10 minutes.
  + 10-15 : This group got 14 responses, accounting for around 47% (nearly half) of my stakeholder sample. Therefore, I can conclude that around half of my end consumers will be running the code for 10 to 15 minutes.
  + 15-20 : This group got 3 responses, accounting for 10% of my stakeholder sample. Therefore, I can conclude that a small minority of my end consumers will be running the code for 10 to 15 minutes.
  + 20+ : This group only got a single response, accounting for around 3% of my stakeholder sample. Therefore, I can conclude that only a very small portion of the end consumers will be running the code for 10 to 15 minutes.
* Overall, I used these results to conclude that most of my end consumers will play the game for around 5 to 15 minutes. Therefore, whilst the focus of the game will be session frequency rather than length, I still need to ensure that the game will engage its consumers for that full length of time as to attract the largest portion of the target audience.
* Each category represents a category of feedback given, among the 30 responses, there were 5 main answers.
  + Category 1: This category represented those who wanted game modifiers (such as speed, difficulty etc.) to vary the experience. This group got 10 responses, accounting for 33.% of my stakeholder sample. Therefore, I can conclude that a significant portion of my end consumers would prefer game modifiers.
  + Category 2: This category represented those who wanted unlockable to work towards (such as speed, difficulty etc.) to vary the experience. This group only got 5 responses, 3 of which noted that this still would not very important to them, and it accounted for 17% of my stakeholder sample. Therefore, I can conclude that a much smaller and indifferent portion of my end consumers would prefer game modifiers.
  + Category 3: This category represented those who prioritised base gameplay over variations. This group got 15 responses, accounting for 50% of my stakeholder sample. Therefore, I can conclude that around half of my end consumers would prefer solid base gameplay mechanics over extra features or modes.
* Overall, I used these results to conclude that I should prioritise base gameplay along with a few modifiers to maximise the overall user satisfaction. Therefore, unlockable currently are not of a high priority as no many would react particularly well to it.

### Stakeholder further contact:

After the initial survey, for each prototype I plan to send over sections of code to be ran (source code?). This would include asking for criticisms including strengths, weaknesses and further improvements/ adjustments. I would then necessarily adjust the project to better fit the end user requirements.

## Problem research :

### Previous solutions:

A multitude of games, of this genre/type have been made before. Some were online flash games and others were mobile, I spent a lot of time on them as a preteen in primary and secondary school. I found these games to attract younger players and were easy to pick up and play a few times before getting bored and finding something else. They also had progression systems- in the form of high scores and leader boards- to keep the user coming back to it every so often as to increase total playing hours.

However, where I feel these games fell short- particularly when attracting larger, older audiences- was not the cartoony style or simplistic graphics but the lack of required timing and skill. The learning curve was too steep. In other words, the games were too quick to get good at. I am not referring to easy to learn controls, but a levelling off in progression that came too early.

Another flaw of these type of games was the lack of variation. This could have been addressed with the game being played different locations with unique characteristics; different equipment with unique attributes; or settings/modes to change how the game plays (e.g., gravity modifiers, enemy frequency, ‘no-enemy’ modes, ‘flipped gravity modes’, faster suits, etc.). This led to them getting repetitive even faster. I found my average playtime of these games to be only around an hour in total (in total, not per session). Therefore, this meant they were played a lot less as they quickly got boring for the user.

On the other hand, one area these games were strong in was the learning process. This is because the control schemes of each game followed general patterns and mimicked each other. This allowed those who were familiar with the genre to use their intuition and experience with it, along with the control scheme guide, to quickly figure out controls. That in turn speeds up learning the controls but also getting used to them as the user is less dependent on the guide.

### Solution essential features:

As mentioned before, timing and skill were something I felt other games lacked. Therefore, the focus of my project is not enemy variety or lots of features, but the core game mechanics themselves. Examples of these include running, jumping, platform generation, obstacle traversal and physics models. This would include things like having to time jumps so that the character does not have to slow down for too long to build up jumping force but also reaches the next edge without stumbling on the corner of it etc. This would address an older player’s fundamental issues with other games of this genre. This is the lack of timing and skill required to ‘get good’. Fixing that issue along with the high score and leader board system, could vastly increase playing hours whilst also expanding the audience.

To tackle the variation problem I previously mentioned, my project would include gameplay modifiers, examples would include different setting and modes such as higher/lower difficulties or higher gravity modes etc. This feature inherently creates new challenges for the player as it would demand different tactics and skills than the base game. Several combinations of game modifiers would allow for smaller, individual skills to be mastered (e.g., learning to not only get good at the base game, but also being good at doing that with half gravity and faster enemies).

One way I could build upon the strong points of previous games, that I found in my research, would be to use to a control scheme as generic, and similiar to others of the genre, as possible. This would greatly reduce the time taken to become proficient with the controls. Therefore, the proportion of players that get bored of adjusting to the game’s mechanics, would be reduced, therefore increasing the player base.

### Solution limitations:

This game is based on a quick ‘pick up and play’ design and getting the user to play lots of sessions is also a high priority. For this to be possible, session duration had to be sacrificed, Therefore, the game is made to played in lots of short durations. There are many successful games out there with a similiar style, for example many mobile games. This limits the skill progression of the game as players will not be engaged for long enough to play for more than five to ten minutes. Playing less, but larger sessions, rather than more, shorter sessions, is better for improving at a game. Consequently, player skill progression will be limited by the game design. However, this problem is partly countered by the variation element to the project previously mentioned. This is because that feature presents new skills to be learned and variations to be mastered, which would drastically increase session length. Also, skill progression would still be developed by the variations as they are derived from the base game.

## Proposed solution:

### Solution requirements:

#### Final solution:

**Features to be included in final solution:**

A main menu. This will be a GUI with links (in the form of buttons) to the main settings menu, leader board tables, high score tables, the credits, and the main game itself. Essentially the main menu will be a user-friendly navigation page to allow quick access to every feature of the project. Every other section will also have links back to the main menu in the form of main menu buttons in the corner. Therefore, the main menu has access to and from every section of the whole program, providing quick and user-friendly navigation.

A sign in and sign-up menu. This will allow users to create, and sign into, their own user accounts. Those with user accounts can access their personal high score table and be included in the leader board table.

A main settings menu. This would include the normal settings, this includes the brightness, access to the credits, proper credit for any licensed material etc. Will also include a ‘game modifier’ sub-section which includes options to select options to change how the main game plays.

The game modifiers menu is a sub-section of the main settings menu. It will include options such as difficulty, gravity, enemy count, platform count modifiers, along with location and difficulty selectors. This will allow me to implement the variation element to the project I previously mentioned.

A leader board table which displays the top 20 playthroughs ever done at the time, ranked by score. This will include run length, distance and score. This will allow competitive users to compete and try beat each other’s scores. (will have separate leader boards for every combination of game modifiers). There would also be a personal high score table which displays the top 10 playthroughs ranked on score that were done on that account. This will include run length, distance and score. This will allow players to try beat their previous high scores and get better. (will have separate leader boards for every combination of game modifiers)

Credits would also be needed to reference of others’ resources that I used in the game. This would include soundtracks, sprites, and artwork etc.

**Solution limitations:**

If I had more time and/or a bigger budget, I could have implemented more features to the project to expand it. This would have allowed an increase in average session length and overall play time.

Other progression systems such as a store, that uses in game currency or an experience system, to unlock new locations, equipment, characters, abilities etc. I could have also used this opportunity to implement in-game purchases into store to allow users to spend real money to boost progression. Advertisements with rewards for watching (e.g., ‘free revival/ second chance on current run).], would have also allowed the user to boost their progression.

With more time I could have animated my own sprites and backgrounds which would have been much more fluid and would have fit better with momentum based in-game physics. With a larger budget, I would have been able to buy rights to better sound effects, sprites, soundtracks etc.

### Solution success criteria:

The main menu must have easy-to-use GUI button links to the main settings menu, account menu, scores menu and credits. This can be tested by asking tester to rate how easy they felt the menus were to navigate.

The account (sign-in + sign-up) menu must allow users to sign into and out of their accounts, as well as set them up, without any errors.

The main settings menu must have GUI multi choice selectors for options including volume, brightness and language. Also, must have a sub-menu for Game modifiers.

The game modifiers menu must have GUI multi choice selectors to difficulty, gravity, enemy occurrence and enemy variation .

The scores menu (Leader board + Personal high score menu) must have GUI button links to leader board + personal high scores. Each table must show the top 10 scores, each showing run length, run distance, score and enemies defeated.

The credits must display references to others’ materials as well as the owner’s name for credit. This can be tested by asking each of the referenced authors how well they felt their work was credited.

Difficulty options for gameplay modifiers must cover a suitable range of skill levels. This will be tested by asking testers to comment on difficulty and if the available options are adequate.

The game modifiers must vary the game enough to be worth changing. This will be tested by asking testers if to rate how much of a difference the modifiers made.

The base gameplay itself must be fun and progressive, rewarding skill and timing. This will be tested by asking testers to rate the gameplay.

# Initial design:

## Problem decomposition:

* The prototype has 6 main sections that are further divided into sub-sections:
  + The main game loop: This is essentially the actual game section. It is the section of code that needs to control the gameplay side of the overall project. The main game loop should be further divided:
    - Calculating sprite positions: This module should calculate when and where the sprites should be drawn based on calculations related to user input and environmental variables.
    - Drawing sprites: This module needs to decide which sprite, from the sprite sheet, to draw to the screen. It should also use the sprite position calculations previously mentioned to draw them at the correct point in time, and coordinates.
    - Drawing background: Free background artwork needs to be drawn onto screen. It also needs move the background left/right at different speeds, looping over itself based on the player’s movement too.
  + The main setting menu: The user needs to be able to adjust a few aspects of the game’s interface. This will include accessibility features, brightness, and audio options (such as music and sfx volumes).
    - The usability/accessibility section: Those with certain disabilities will need the game to be tweaked for them to be able to play. Therefore, there will need to be features that make the game accessible to them and change it as to make sure given conditions do not hinder their game experience.
  + The leader boards/ high score section: Data needs to be saved from various playthroughs. This is necessary to allow the user to compete with other players, or their own scores. This is critical to increasing playing hours as mentioned in the analysis section.
  + The account system: This needs to manage the users’ account data as well as handle the signing up and signing in procedures.
  + The Game database: This will simply store all game-related data in several databases. Each account should have its account details, personal high scores and leader board entries stored here.

## Solution description:

### Solution structure.

* The prototype solution has 6 main sections that solve the sub problems previously mentioned:
  + The main game loop: This is essentially the actual game section. It is the section of code that will control the gameplay side of the overall project. The main game loop will be further divided:
    - Calculating sprite positions: This module will calculate when and where the sprites should be drawn based on calculations related to user input and environmental variables.
    - Drawing sprites: This module decides which sprite, from the sprite sheet, to draw to the screen. It should also use the sprite position calculations previously mentioned to draw them at the correct point in time, and coordinates.
    - Drawing background: This module will draw the selected background artwork. It will also move the background left/right at different speeds, looping over itself based on the player’s movement too.
  + The main settings menu: This module will allow the user to adjust the brightness, sound effects volume and BGM volume. This will also include accessibility features, brightness, and audio options (such as music and sfx volumes).
    - The usability/accessibility settings section: This will tweak the game to make it more friendly for colour blind players. Therefore, there needs to be a colour-blind mode that adjusts all the GUI RGB values to allow colour blind players to play the game unhindered.
  + The leader boards/ high score saving section: This will save the data from various playthroughs. This allows the user to compete with other players, or their own scores. This is will increase playing hours by giving the player base something to work towards.
  + The account system: This will manage the users’ account data as well and the signing up and signing in procedures.
  + The Game database: This will consist of several databases used to store data to run various features such as the leader board and personal high score table. Therefore, each account will have its assigned account details, personal high scores and leader board entries stored here.

### Inputs, processes, outputs and storage.

|  |  |  |
| --- | --- | --- |
| **Input:** | **Processing:** | **Output:** |
| Press menu buttons: | The buttons are links that load different pages. Button presses will be retrieved as user inputs. Clicking them will retrieve and load the page that link represents. For example, if the player is one the main menu and clicks the options button, the options menu is loaded up and | Display respective page. |
| Move GUI sliders. | The options sub-menus involve sliders to adjust different variables such as brightness and audio volume. Each slider will be composed of 5 sub-sections, at the end of each the slider can be placed. This will select that value for that setting. For example, for volume this would be moving the volume up 20, 40, 60, 80 and 100%. Moving the slider will be retrieved as the user repeatedly pressing the mouse (holding it down) on the slider button coordinates and moving it. | Adjust respective setting by a fifth. |
| Click type boxes. | Clicking the type box will essentially activate it. This input will be retrieved as the user pressing one of the character keys. Now entering a character key will type that into that entry. For example, normally pressing the ‘A’ key would not do anything. However, after activating the type box, the character ‘A’ would be typed into that entry. | Activate type box. |
| Character keyboard press (once type box is activated). | Once the type box in activated, pressing character keys (includes upper and lowercase alphabet, numbers and punctuation) will type that key to the entry. This input will be retrieved as a mouse click on the GUI button coordinates. | Type key to entry. |
|  |  |  |

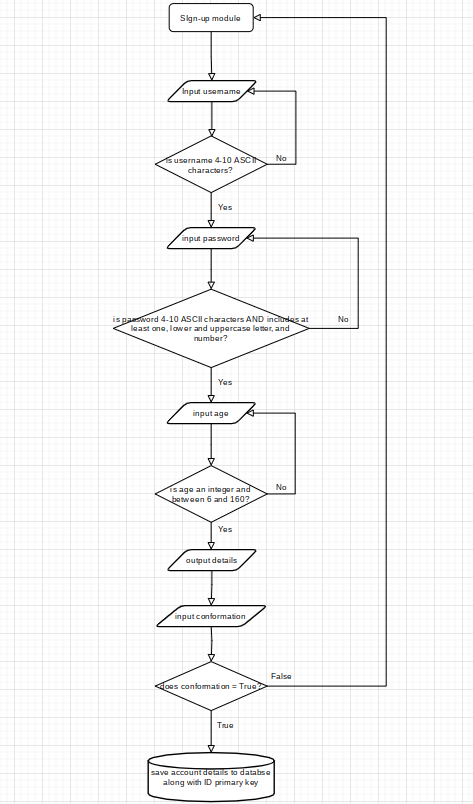
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data storage table:** | | | | | |
| **Class:** | **Attribute:** | **Purpose:** | **Data type:** | **Validation:** | **Variables involved.** |
| Sign up | Username | Stores user’s in-game name | String | Validate username by making sure it is 4-10 ASCII characters. | Username (str)  Character (str)  ASC\_V (int),  Validity (Bool) |
| Age | Stores user’s age | Integer | Validate age by making sure it is an integer from 8-120. | Age (int)  Type (str)  Validity (bool) |
| Password | Stores user’s security password | String | Validates password by making sure it is 4-10 ASCII characters. Further validation making sure at least 1 letter, upper and lowercase, and 1 number is included. | Password (str)  Character (str)  ASC\_V (int)  Validity (bool) |
| details | Combines all details and is used for user confirmation of details | Array | Double checks details with user and asks them to confirm account creation. | Username (str)  Age (int)  Password (str)  Confirmation (bool)  Account ID (int) |
| Sign in | details | Checks the username of the account the user is trying to sign in to, to the existing database usernames. This is so that the matching account can have it’s password compared, deciding whether the user is granted access to that account. | Array | Checks if username is on database, if not alerts user, if it is, checks password matches corresponding database password. | Username (str)  Password (str)  DB\_name (str)  DB\_ID (int)  Permission (bool)  Account\_ID (int) |

### Algorithms:

Here are the flowcharts for a few modules of the project:

#### Sign up module:

(insert flowchart screenshot here:)

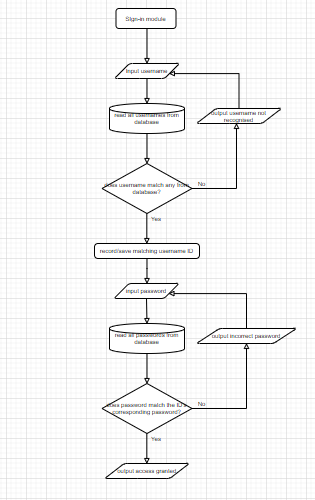


(insert description/explanation below:)

* This is the flowchart for the sign-up module.
* The ‘username’ input will allow the user to input the username they want for their account.
* The first decision will check that the username is an appropriate length and does not contain any non-ASCII characters ((will later implement check to see if username is already taken to increases robustness)).
* The ‘password’ input will allow the user to input the password they want assigned to the account they are creating.
* The second decision will check that the password has at least 1 upper and lowercase letter, along with at least one number. This is to make the password more secure, making the account harder to illegitimately access. This decision will also include a length check to ensure that the password is an appropriate length.
* The last detail to assign the account will be age. The age of the user will not be publicly displayed on leader boards or used for security checks; it will be more for statistical purposes. For now, however, this is a required field ((will later implement this as an optional field)).
* There will be a check for the age that ensures it is the correct data type and a reasonable number. This will make sure invalid options are not entered and things like averages of the ages can be measured for statistics to help with development.
* The details will be output for the user to double check that they are correct.
* The confirmation input will check that the user is/is not okay with them.
* The conformation decision will decide whether to save the account to the database or loop back to the beginning to restart the account creation ((Will later add option for user to change mind and cancel account creation altogether.)).
* Lastly, the details will be added as a new entry to the account database with its own ID primary key.

#### Sign in module:

(insert flowchart screenshot here)



(insert description/explanation below)

* This is the flowchart for the sign-in module
* The ‘username’ input will allow the user to input the username of the account they want access to.
* The database procedure will then check the entered username against all the others in the database. This will be done via linear search.
* The decision will then check whether this username matches any of the others:
  + If it does not, then the program will output to the user that the username is not recognised, and the user will be asked to input another. This is to ensure that the user is logging into an account that actually exists as to avoid problems later.
  + If it does, the program will save the username’s corresponding ID number. This will allow for comparison of other details later, as the program will know which account the user is trying to access.
* The ‘password’ input will allow the user to input the password security of the account they want access to.
* The password is then checked against the corresponding password of the ID number saved earlier. Now that the ID has been saved, the program does not have to go through the whole database and can straight away compare the entered password to the account password.
* The decision will then decide what to do based on whether the passwords match. This is a security check to make sure that the user has legitimate access to that account:
  + If they do not, the program will output to the user that the entered password was incorrect and will then give them another attempt to enter the correct password ((Will later implement passwords attempt limit to prevent brute force attacks by locking the user out after a given number of failed password attempts.))
  + If they do, the user will be granted access to the account and a confirmation message will appear.

#### Leader board module:

(insert flowchart screenshot here)

Diagram

Description automatically generated

(insert description/explanation below)

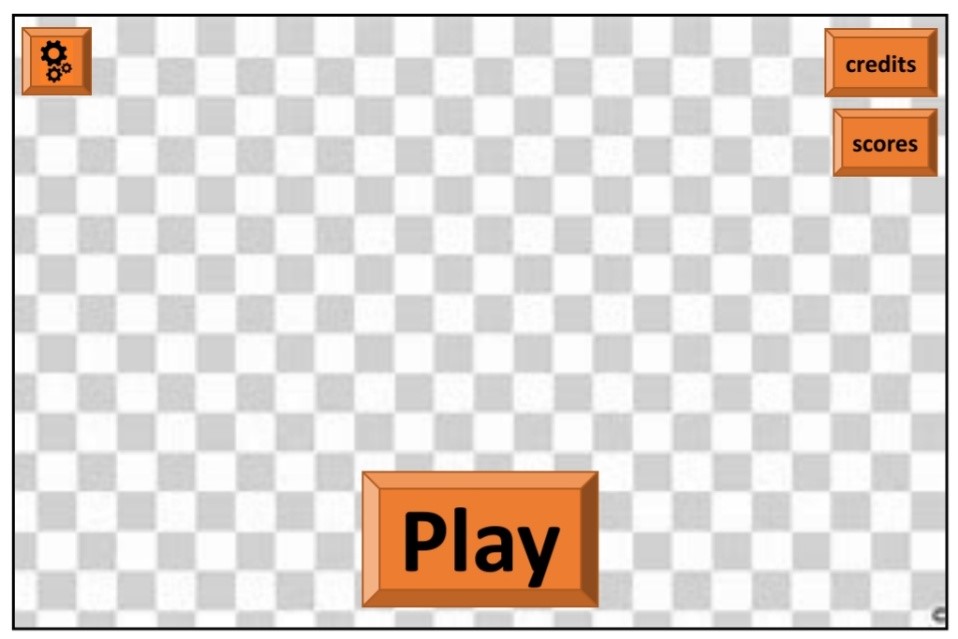
* This is the flowchart for the leader board module.
* This is the ‘account check’ section. The first decision will check whether the user is logged into an account or not. This will be decided by a Boolean variable extracted from another algorithm in charge of logging in:
  + If False, the program will deny access to saving details to the leader board, and output that an account is required to do so. It will then skip over that section, to the ‘display section’.
  + If True, the program will proceed to the ‘consent’ input in which the user either gives or denies consent to have their details saved to the global leader board for everyone to see.
* This is the ‘consent’ section. The ‘consent’ input will produce a Boolean value representing whether the user wants to be a part of the leader board or not.
* The next decision will check whether the user wants to be on the leader board. This will be determined by the Boolean output of the consent section was ‘True’ or ‘False’:
  + If False, the user will not be added to the leader board and the program will skip over to the ‘display’ section.
  + If True, the user’s username and high score will be added to the database, along with a primary key ID number (ordered by when that entry was last added/updated). The entry would be added to the end, after which the database would be sorted by high score, from high to low.
* Next up is the ‘display’ section. Firstly, there is a ‘display’ input which checks whether to user wants the leader board to be displayed, which will return a Boolean value.
* The next decision checks whether the obtained Boolean variable is set to ‘True’ or ‘False’:
  + If False, the algorithm skips to the end and is terminated.
  + If True, the algorithm will run another ‘display leader board’ sub routine.

### Usability features:

Here are some designs for the initial prototype GUI:

#### Main menu:

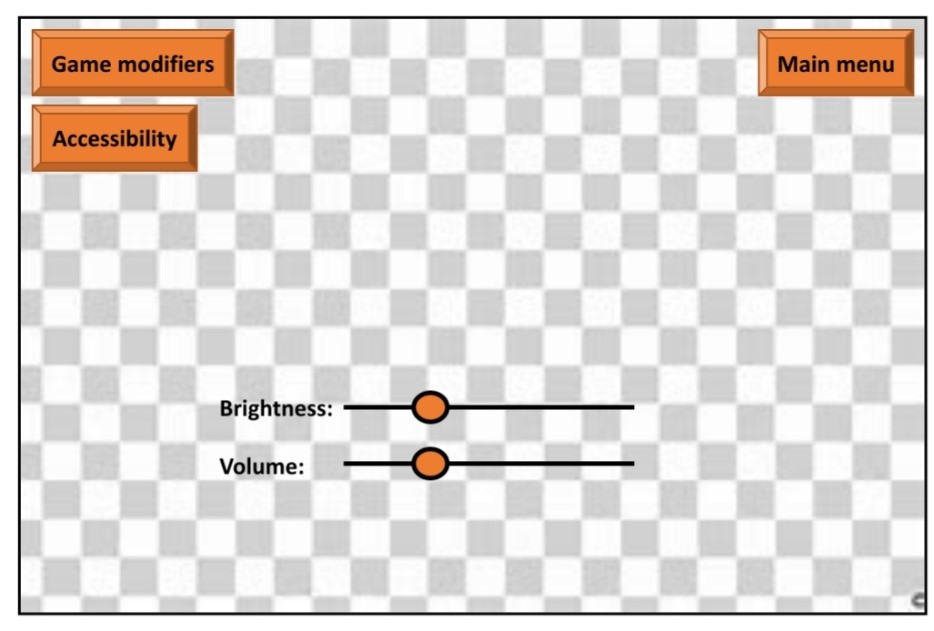
* This is for the main menu:



* As with most GUIs the main acts as a homepage and starting point for the user, for this reason it has links to the other main aspects of the game which In turn have access to their respective sub-components.
* This increases the main menu’s effectiveness as it is either directly or indirectly connected to every other GUI page which makes it an effective homepage.
* Another key feature is that the main menu will be the only place the user can start the main game. This reinforces it’s role as the homepage as it gives exclusive direct access to the other main component of the overall program, the game itself.

#### Main settings menu:

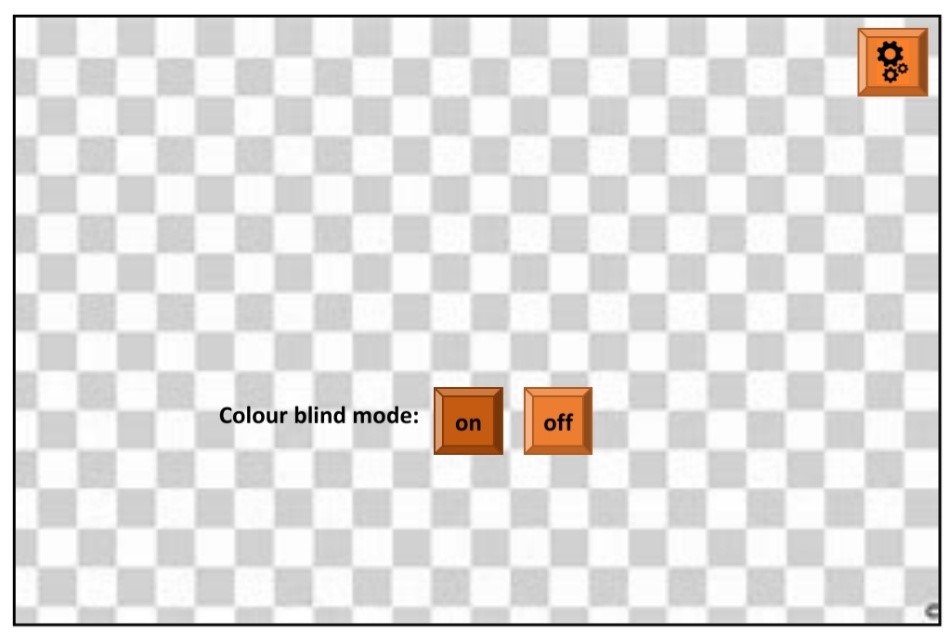
* This is for the main settings menu:



* The main settings menu is initially accessed via it’s link on the main menu page.
* From here, there are also links to it’s two sub-menus, the game modifiers settings menu and the accessibility settings menu.
* The main menu button is used to traverse back to the main menu, allowing access to the other GUI pages the main menu hosts links to, as well as the main game.
* The brightness and volume settings will implement sliders, this will help users select a very specific value for these two settings out of a gradient of distinct intensities.
* This will help the user more specifically tailor the GUI and game visual and audio display to their situation.

#### Accessibility settings menu:

* This is for the accessibility menu:



* The accessibility settings menu is initially accessed via it’s link on the main settings menu page.
* The main settings menu button is used to traverse back to the main settings menu, allowing access to the main menu hosts and the game modifier settings menu.
* One common problem colour blind users have with many GUI’s is that they often have wide range of colours used, increasing the chances of confusing colours for each other, as there are more colours to process and navigate through, providing challenges for colour blind players.
* For this reason I will limit the colour palette used for the game display; the fewer colours I use in my design, the fewer instances there will be for confusion.
* This minimalistic design is a timeless and will help when designing for colour accessibility.

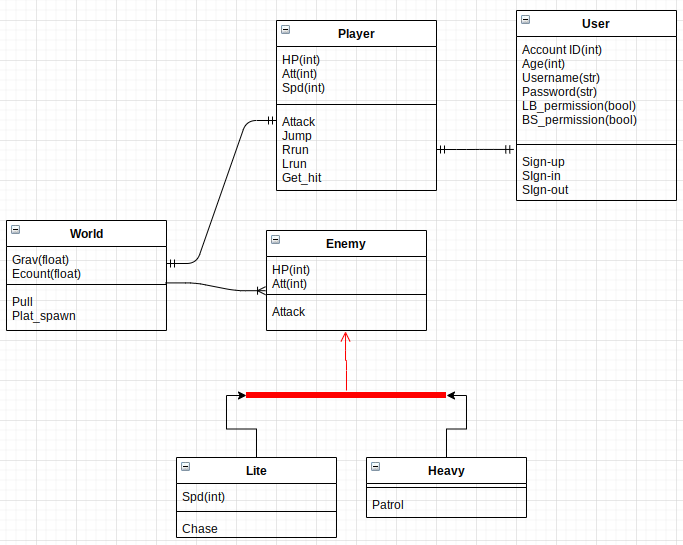
### Key variables and structures:

|  |  |  |
| --- | --- | --- |
| **Variable table:** | | |
| **Key variable name:** | **Purpose:** |
| Username | This variable keeps track of the user’s account name. It is used as reference in the global leader board tables, to validate login attempts, and to help the user recognise their account. |
| Age | This variable keeps track of the user’s age if they have an account. This will not publicly be displayed however it will still be a part of the sign-up process and the account database for statistical purposes. |
| Password | This variable keeps track of the user’s account password. This will be used exclusively for security and to validate login attempts. For this reason, it will also be kept in the account database. |
| Validity | This is a simple Boolean variable that will be used for various validation checks. Examples of these would be checking entered account details, for the sign-in and sign-up modules. This will include making sure a reasonable age is entered (integer between 1 and 150) and that username and passwords are an appropriate length. |
| Account ID | This variable will be used as a primary key for the account database. It will also be used by programs to check which account a user is trying to access so it can verify if they have the necessary authorisation. |
| Conformation | This is a simple Boolean variable that will be used for various user confirmations. Examples of these include account creation confirmation and leader board participation confirmation. |
| Highscore | This variable will store the user’s last saved high score. It will be used for the global leader board to allow players to compete with each other. |
| Best\_score | This variable will be an array storing the user’s top 10 saved scores for the personal high score table. |
| Permission | This is a simple Boolean variable that will be store whether the user has given permission for the program to do something. This will be used in various modules, such as the leader\_board module, where it would confirm that the user wants to participate in it. |
| G\_multiplier | This is a multiplier that will be used in the game modifiers module to represent how strong the user would like the gravity to be. |
| Enemy\_count | This is a multiplier that will be used in the game modifiers module to represent how often enemies appear. |
| Volume | This is a multiplier that will be used in the settings module to represent how loud the user wants the overall game to be. |
| Bright | This is a multiplier that will be used in the settings module to represent how bright the user would like the GUI to be. |
| Screen size | This will store the screen size constant to be referenced when drawing the menus ,character sprites, background art, platforms, and projectiles. |
| Key\_inp | This variable will be in the game loop module and will store the user’ current input within that time frame (60th of a second). |

|  |  |
| --- | --- |
| **Validation table:** | |
| **Variables involved:** | **Validation checks:** |
| Username (str)  Character (str)  ASC\_V (int),  Validity (Bool) | * Validates username by making sure it is 4-10 ASCII characters |
| Age (int)  Type (str)  Validity (bool) | * Validates age by making sure it is an integer from 8-120 |
| Password (str)  Character (str)  ASC\_V (int)  Validity (bool) | * Validates password by making sure it is 4-10 ASCII characters * Further validation making sure at least 1 letter, upper and lowercase, and 1 number are included |
| Username (str)  Age (int)  Password (str)  Confirmation (bool)  Account ID (int) | * Enters details * Double checks details with user and asks them to confirm account creation |
| Username (str)  Age (int)  Password (str)  DB\_name (str)  DB\_ID (int)  DB\_passW (str)  Permission (bool)  Account\_ID (int) | * Checks if username is on database, if not alerts user * If is, checks password matches corresponding database password |
| Username (str)  Age (int)  Password (string)  Account\_ID (int)  Highscore (int)  Best\_scores (arr) | * Will store user data across a single table * Can be added to, removed from, and accessed by various other modules, * Does not ever output age, age is only for statistical purposes * Will use linear search algorithm to traverse database * Will list top 10 saved scores as a GUI table |
| Username (str)  Highscore (int)  LB\_pos (int)  LB\_consent (bool) | * If LB\_consent is true, will automatically add user’s best score to the leaderboard and update it if it gets better * Does not shows age or user ID * Will also be displayed as GUI table |
| On (bool)  R\_change (int)  B\_ change (int)  G\_change (int)  Message (str) | * Will appropriately change RGB values to tailor towards colour-blind * Will print conformation message specifying what kind of colour-blind condition the mode suits * Will also refer to wiki links for more info |
| G\_mulitplier (float)  Enemy\_count (int) | * Will multiply gravity strength by selected multiplier * Change enemy frequency based on the values they input (stored as integer representing average enemies per 10s) |
| Volume (int) | * Will adjust game volume based on a percentage entered by the user |
| Bright (int) | * Will adjust game RGB values based on a percentage entered by the user |
| Art\_link (link)  Screen\_size (int) | * Will draw image in link onto background to find window size |
| Key\_inp (str)  Player\_pos (arr)  Pos\_change (int) | * Will factor in keyboard inputs (jump, run, crouch etc.), collisions (knockbacks) and footing (standing on ground or mid-air) to calculate changes in the player’s sprite's position. |
| Player\_pos (arr)  Sprite\_link (link) | * Will render player sprite wherever the player position is |

#### Classes:

(Insert class diagram here:)



(insert description/explanation:)

* Player class: This class represents the player’s character:
  + The attributes include hit points, attack, and speed. These are set by default to be the same for every player.
  + However, getting certain high scores will unlock upgrades for each of these attributes. The methods include attacking, jumping, running left and right, and getting hit:
    - The attack method will allow the player to do melee attacks to damage enemies and kill them if their hit points are lowered to 0 or below.
    - The run left/right method will allow the user to move the player left or right.
    - Jumping will allow the player to jump at the cost of losing speed and momentum.
    - Getting hit will reduce HP and lose the player speed and momentum.
  + This class will have a one-to-one relationship with the user class as each user has only one assigned player in the world.
  + This class will have a one-to-one relationship with the world class. This is because the in-game world only has one playable character.
* World class: This class represents the in-game world:
  + The attributes include the gravity multiplier and the enemy count multiplier.

### Testing approach.

* The prototype has 6 main sections that will need testing:
  + The main game loop:
    - Calculating sprite positions:

|  |  |  |
| --- | --- | --- |
| **Calculating sprite position module alpha test table:** | | |
| **Test data:** | **Relation to success criteria:** | **Explanation:** |
| Left arrow key | Valid input to x movement. | Pressing the left arrow key should decrease the x coordinates of the player. This is to test that the inputs produce the correct change in the x position. |
| Up arrow key | Valid input to y movement. | Pressing the up-arrow key should cause the player character to jump. This is to test that the inputs produce the correct change in the y direction. |
| ‘F’ key | Valid attack input. | Pressing the ‘f’ key should produce a melee attack from |
| ‘K’ key | Invalid key input. | Pressing this key should do nothing and not affect the game. |
| ‘4’ key | Number key input | Pressing a number key should do nothing and not affect the game. |

* + The main setting menu:

|  |  |  |
| --- | --- | --- |
| **Main settings menu module alpha test table:** | | |
| **Test data:** | **Relation to success criteria:** | **Explanation:** |
| Click button | Valid input to GUI navigation | Clicking the mouse whilst hovering over the GUI buttons should activate the link associated with them and load up that page. For example, the main settings button would load up the main settings menu page. |
| Click away from button | Invalid input to GUI navigation | Clicking the mouse whilst not hovering over a GUI link should do nothing, however the timing and coordinates of the click should still be accurately recorded. |
| ‘F’ key | Invalid input to GUI navigation | Pressing this key should not do anything as that input is not associated with any links/actions |
| Drag slider | Valid input to setting selection | Holding and moving the mouse over the slider should move it along to the next node along the scale and correctly adjust it to that setting. For example, moving the volume slider left should decrease volume. |

* + - The usability/accessibility section:

|  |  |  |
| --- | --- | --- |
| **Accessibility menu module alpha test table:** | | |
| **Test data:** | **Relation to success criteria:** | **Explanation:** |
| Click GUI button | Valid input to setting selection | Clicking the ‘on’ button for colour blind mode should instantly switch up the RGB colour scheme to make it more accessible for colour-bling players. Clicking the ‘off’ option should turn it off the return the user to the normal setting. |

|  |  |  |
| --- | --- | --- |
| **Sign up module alpha robustness test table – password creation:** | | |
| **Test data:** | **Relation to success criteria:** | **Explanation:** |
| (null) | Null | This is testing the null password validation. it should return the password as invalid as null values shouldn’t be allowed for passwords. |
| ‘lo£p2rski3%’ | Invalid | This is testing the character count range validation. It should also return this password as invalid as the maximum character count is 10, not 11. |
| ‘lopperskis’ | Invalid | This is testing the number and symbol check. It should also return this password as invalid as the minimum single number and symbol haven’t been included. |
| ‘34^\*£(20’ | Invalid | This is testing the letter check. It should also return this password as invalid as the minimum single letter hasn’t been included. |
| ‘Xe1\*f2’ | valid | This is testing whether the checks accept valid passwords. This should be returned as valid as it meets the password requirements |

# Development:

## First prototype:

### Analysis.

For the initial prototype of the project, I will focus on creating a functioning iteration. The iteration I started with was the main menu module. The main menu module was a fitting first iteration as it acts as the iterative base and start point of the game. To make the main menu buttons however, I found that it was more efficient to have a buttons module that I could call button classes from.

### Design.

The current algorithm does not involve much input or decision as user input has not yet been implemented. However, to make a functioning iteration there has to be an end condition. Therefore, the first user input I decided to implement was the overall end condition of the game: the input of attempting to close the game window.

Here is a basic flowchart showing the loop end condition:

Diagram

Description automatically generated

### Test data.

As there is only one possible input as of yet, the only available test data is to close the program and see if the program responds as it should. However, it will still be necessary to test the programs I have written so far to see that they work as intended, regardless of a lack of test data.

|  |  |  |  |
| --- | --- | --- | --- |
| Test data type: | Test data: | Expected output: | Justification: |
| Valid. | Exit program. | Program window closes and program ends. | This test data will show whether the procedure of quitting the game works as intended. |

### Implementation.

#### Quitting procedure.

Here is the implementation of the quitting procedure. I created the ‘running’ variable. This keeps track of whether the user wants the program to keep running or not. It is set to ‘True’ by default.

Text

Description automatically generated

Here I implemented the ‘running’ variable as the iteration condition for the main menu loop. This allows the game loop to only stop when the user wants it to, but still ensure that it runs otherwise.

Graphical user interface, text

Description automatically generated

Here I allowed the user to set the ‘running’ variable to False by clicking on the exit button of the game window, this activates the game loop end condition which stops it from iterating. This input is printed/logged as “the user asking to quit.” As that is what the program assumes their intention to be when entering this input. This section is inspired by syntax from a stack overflow solution [1].

Text

Description automatically generated

This section of code is executed when the condition, for the main loop to stop, is met. It prints that the program has been terminated. The time sleep keeps the program open for a short time after closing the game window to give the user time to see this message before it also terminates.

Graphical user interface, text

Description automatically generated

#### Background drawing:

Here is the implementation of the background drawing:

Text

Description automatically generated

Each building Is drawn separately as a black rectangle, with the moon as a light grey circle. These are drawn at the start of the drawing section to ensure that they don’t block off, and interfere with, anything else. The ‘constants’ module has been referenced to retrieve the desired RGB colour values and screen height.

‘x’ saves the position of the first building and is used as a reference, along with the previous buildings’ cumulative widths, to position each building along the x axis. The y position of the buildings is altered based on the screen height – the building height to make sure that there’s no gap between them and the bottom of the screen, but that the whole building is shown on screen.

#### Rendering text:

Here is the implementation of rendering text to the screen for the title and buttons. This section is inspired by syntax from a stack overflow solution [2] :

Text

Description automatically generated



Firstly, I saved the file name of the font I wanted to a variable. Next, I created a font based off the font file referenced earlier and saved that to a variable. Then the text was rendered with the font variable and the text Rect was called to position it. The last step was then to blitz the text and text Rect variables to the screen.

### Testing:

|  |  |  |  |
| --- | --- | --- | --- |
| Test data type: | Test data: | Expected output: | Actual output: |
| Valid. | Exit program. | Program window closes and program ends. | The program closed the window and was terminated as expected. |

### Remedial actions.

While developing the initial prototype main menu I ran into errors:

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Problem screenshot/s + description | Solution screenshot/s + description | Solution justification |
| 1 | Text  Description automatically generated  Text  Description automatically generated   * Here, during each iteration I saved/updated each building variable which was inefficient and unnecessary. | Text  Description automatically generated   * The variables were only there to indicate what that section of code did, therefore it was more efficient to simply have a comment there which could be ignored by the interpreter. | * This saves memory; more importantly however it reduces overhead and saves time for every iteration, which is set at 60 times per second. |
| 2 | Whilst trying to reference constants from my constant file, I found that the file could not read the code that dictated the constant values as they required the pygame library to be read.  Text  Description automatically generated | I added a line to the constants file to import the pygame library. | * Adding this line allowed the interpreter to read the pygame syntax and assign the correct value to the background variable, allowing the main menu module to correctly reference and use it. |

### Evaluation.

* Buttons do not yet do anything. I need to implement links to the other game modules when the mouse is pressed on the button coordinates are clicked. Visual cues for clicking and hovering are also needed to clarify the input to the user.
* The options and scores buttons have not been implemented yet. It is important that these two are added. The main menu is supposed to directly or indirectly provide access to every part of the overall game which currently is not the case. Therefore, buttons with links to other pages need to be implemented.
* There have been deviations from my original solution structure:
  + Now there is a ‘buttons’ module. This was more efficient and far more readable than defining each button class within the main menu module. Another benefit is that this module can now be re-used in the other modules without confusing the developer (e.g. if the ‘options menu’ module imports the ‘main menu module text button class’). This module could also be re-used in developing other GUI’s for other projects too.
  + There is also a ‘constants’ module. This was again more efficient than defining various constants (such as colour RGB values and fonts) within each module. There is also ample opportunity to re-use this code in other modules and future projects too.

#### Stakeholder contact:

I have not sent this prototype to any of my stakeholder sample. My reason for not doing this is that the prototype is too small scale and that, not only is the gameplay missing, but the GUI is also just starting out in development. Therefore, I concluded that I will need to create more developed prototypes to send off to the stakeholder.

## Second prototype:

### Analysis.

The previous prototype had two main flaws. A lack of content to be able to get stakeholder to test, and a lack of structure.

* The lack of content was due to time restrictions in which to develop the initial prototype. Due to the time shortage I had to make a smaller, less time-consuming prototype. For this reason, I was only able to develop a main menu with a single functionless button.
  + Therefore, this prototype will be larger and more functional. It will also include the main settings and accessibility settings menus. This will allow for stakeholder feedback, as I will have a product with enough content for stakeholders to critique.
* The lack of structure was due to poor planning. One area in particular that suffered from this was the main menu game loop. The rendering, calculation and event-retrieving sections of code were all mixed together, resulting in spaghetti code with no sense of direction or clarity, even at a small scale this reduced the code’s readability.
  + To solve this problem, I will implement a structure into the game’s constituent game loops. The structure I have chosen is inspired from ‘programarcadegames’ [3]:

Text

Description automatically generated

### Design.

* Here is a class diagram for the buttons module:

Diagram

Description automatically generated

* I will also add extra features to the GUI buttons I have implemented so far. This will be to have hover and activate them.
* I will also implement the main settings menu.
* I will need to create links back and forth between the main menu and the main settings menu.

### Test data.

I have been adding temporary print statements in different parts in the code and reading the console (whilst running the code) to see if results meet expectations.

Now there are a total of three possible inputs. Quitting, hovering over a button, and pressing a button. The result of the quitting input (the quitting procedure) has been thoroughly tested in the previous iteration and has remained unchanged.

However, the hovering and pressing input results have not been tested. Therefore, this protoype’s testing will largely be focused on button functionality and transitioning between game menus/sections.

|  |  |  |  |
| --- | --- | --- | --- |
| Test data type: | Test data: | Expected output: | Justification: |
| Valid. | Hover over button | Buttons turns darker colour until mouse is moved away. The button showed only be considered hovered over when the mouse is on the button’s coordinates. | This test data will show whether the program GUI can successfully check whether a button is being hovered over or not. |
| Valid | Hold mouse over button | Buttons turns even darker colour until mouse is released or moved away. Button should only be considered held when mouse is hovering over and being pressed. | This test data will show that the button activation and hover recognition works as intended. |
| Valid. | Activate menu navigation button | The text box should only activate when the button is highlighted and the mouse is pressed, held (even for a very short time), and released without leaving the button. Only after the button releases should the button be activated. If these conditions aren’t met, the button changes colour without being activated | This test data will show whether the procedure of switching between menus works as intended. |

### Implementation.

#### Page layout:

* Here is a diagram representing the layout of the game pages:
* The main menu is the homepage and so it has direct access to other three main components of the GUI: the settings, the account section, and the score ranking menus.
* The main settings menu has links to the accessibility and game modifiers menu as I consider them sub-categories of settings, therefore it makes sense to have them sub-divide from the settings menu.
* The account menu has links to the sign in and sign up menus as they are smaller functions that contribute to the account system’s functionality.
* I chose the leader board menu to be the forefront of the score ranking system. In the current age of networking (and particularly game-related online interaction), competing with others’ scores is their main aim, with besting personal records being an afterthought. I decided therefore that my menu layout should reflect this as I would expect the global leader board to be the more heavily used, important aspect of the score ranking system, with personal bests as an afterthought/additional feature.

#### Mouse:

* There are two aspects of mouse input that this program uses. The status of the left mouse button and the mouse screen coordinates.
* To handle mouse inputs I utilised syntax I found on ‘stack overflow’ (Mouse button detection source: [4]):



* As for the button status I decided I also wanted to implement a timer that recorded the duration of each time the mouse was held down.

#### Buttons:

##### Hover method:

* The buttons module also received two key methods, the hover module and the activated module:
* Both modules took a lot of time to develop. This was because I first tried implementing functions to specifically carry out those for a text button object, the start button, in the main menu module. While this was successful, constantly having to call attributes, getters and setters, as well as the fact that I would have to do this every time I wanted to create a new button and make them interactive.
* Therefore, I instead opted for methods within the button parent class:
  + Here is the hover method:

Text

Description automatically generated

* + What this method does is first check if the mouse is in the x coordinate range of the button, and then the y coordinate range. If both are True, then the variable representing whether the mouse is over the button or not is set to True.
  + To calculate the x coordinate range, the method gets the object’s x coordinate and half of x axis size. The x range is then calculated as from the start which is the half subtracted from the centre, to the end, which is the half added to the centre.
  + This check is done in the activation module every loop to check whether the mouse is over the button at that time, using the mouse x and y position as parameters.

##### Activation method:

* Here is the activation method:

Text

Description automatically generated

* + - What this method does is, using the mouse coordinates and button status, check whether the button has been activated or not. However, it’s also responsible for making sure that the button is the correct colour to indicate it’s status:
      * The button essentially has three colour modes: light orange indicating neutral, medium orange indicating hovering, and dark orange indicating holding.
      * As well as making sure the correct colour is displayed, the button must also make sure that the button is only activated if the mouse was held over and released over the button, without having left those coordinates, that is what line 72-79 makes sure of.
    - This check is done every frame to make sure that every button is always displayed as it should be and that they’re only activated when they are supposed to be.

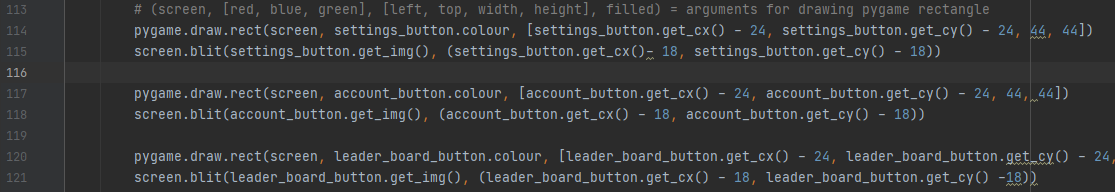
##### Image buttons:

Text

Description automatically generated

Text

Description automatically generated



* For the settings button I used a settings icon that was downloaded from ‘flaticon.com’ , made by ‘Dmitri13’ [5]. (Icon made by Dmitri13 from [www.flaticon.com](http://www.flaticon.com))
* For the leader board button I used a trophy icon that was downloaded from ‘flaticon.com’ , made by ‘Dmitri13’ [5]. (Icon made by Dmitri13 from [www.flaticon.com](http://www.flaticon.com))
* For the account button I used a profile icon that was downloaded from ‘flaticon.com’ , made by ‘Dmitri13’ [5]. (Icon made by Dmitri13 from [www.flaticon.com](http://www.flaticon.com))
* For the home button I used a home icon that was downloaded from ‘flaticon.com’ , made by ‘Dave Gandy’ [6]. (Icon made by Dave Gandy from [www.flaticon.com](http://www.flaticon.com))
* The images buttons worked by firstly inheriting all the attributes and methods from the parent button class. This included the hover, activate, and colour setter methods, along with the attribute getter and setters. Additionally, I added a new setter for the image attribute which got the parameter URL and used the pygame image loading subroutine on it.
* As for actually displaying the buttons, as they were downloaded as 32 by 32 pixels and transparent, no resizing or adjustments were needed, allowing me to simply blit the images onto the game surface. For the coordinates I used trial and error to come up with values that would have the icons appear in the middle of the button and still have the hover/activation coordinate ranges lining up as intended. The values found worked for all buttons as they’re relative the ‘cx’ and ‘cy’ parameters and are the same size.

### Testing.

|  |  |  |  |
| --- | --- | --- | --- |
| Test data type: | Test data: | Expected output: | Actual output: |
| Valid. | Hover over button | Buttons turns darker colour until mouse is moved away. The button showed only be considered hovered over when the mouse is on the button’s coordinates. | The expected output exactly. |
| Valid | Hold mouse over button | Buttons turns even darker colour until mouse is released or moved away. Button should only be considered held when mouse is hovering over and being pressed. | The expected output except that when holding away from the button and then hovering over, the button goes to the even darker colour when it wasn’t intended to do that. That being said, the holding checks work fine and I decided to use them as a heuristic as they still mostly do as intended. |
| Valid. | Activate menu navigation button | The text box should only activate when the button is highlighted and the mouse is pressed, held (even for a very short time), and released without leaving the button. Only after the button releases should the button be activated. If these conditions aren’t met, the button changes colour without being activated | The expected output exactly. |

### Remedial actions.

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Problem screenshot/s + description | Solution screenshot/s + description | Solution justification |
| 1 | Text  Description automatically generated  Here, there would be nothing to reference the x positions. Therefore, the variable would have to be created each time a new menu opened. This resulted in the building positions being set to default, resulting in a jittery and rough transition between menus. | To solve this I passed in x as a parameter from the previous menu. | X is the variable used to keep track of the building coordinates, by passing it as a parameter into the next menu procedure, the menus can transition over with the buildings retaining their exact positions and then continuing to move seamlessly. |
| 2 | Text  Description automatically generated  There was an issue here that reduced the modularity of my project. Each time I wanted to call a menu’s main function after defining it (in order to run in seperately from the rest of the program) this error would come up and stop the program. | Graphical user interface, text, application, chat or text message  Description automatically generated  I found the issue to be that I never initialised pygame within the buttons module. This was solved by adding a statement to initialise pygame | This was because I instead did this is in the main module and therefore never ran into this as I always ran each menu by running the main module and switching over to that menu with the GUI button links. This way I can now run each module even when the links from other menus haven’t yet been implemented or aren’t working. |

### Evaluation.

* Overall, more progress was made with this iteration than expected.
  + Initially, only a few menus were supposed to be developed, instead, I realised the buttons module greatly increased button reusability. Also due to the fixed setup and game loop structure, many sections of code (sometimes being over 150 lines) could be copied and pasted (with minor tweaks) and would run as intended, saving a lot of development time.
  + Therefore, this prototype ended up being much more reusable and time efficient to develop whilst still being fully modular and working as planned and intended. Reflecting on this, for future prototypes, creating modular and reusable code will be a priority to increase efficiency.
* One slight issue with this code is that the buttons don’t work exactly as I intended them to. I noticed that when hovering away from them and then moving the mouse over, the buttons turned dark orange when they were supposed to turn medium orange. However, trying to resolve this was resulting in lots of additional overhead per loop, as well as costing a lot of development time. For this reason, I decided to use this hover/activation setup as a heuristic as it still largely worked exactly as planned.

#### Stakeholder contact.

* For this prototype there was more content. Therefore, there was now enough material for a small sample to actually critique.
* I sent an executable file of my second prototype to seven randomly selected members the stakeholder sample, asking for short reviews, here were the results:
  + ‘The frame rates occasionally drop but this isn’t a major problem’ – This came up in 2 reviews:
    - Adding lines of code to clean up unused variables could maybe help reduce this issue, however, this happens even on the first menu open meaning this issue may be a lot more complex than that.
    - This however was a minor complaint, so perhaps more emphasis should be placed on trying to not let this get worse rather than trying to completely eliminate it altogether.
  + I also asked each tester about the button functionality and highlighting. This didn’t come up naturally in any reviews however:
    - All of the testers replied with the highlighting looking fluid and accurate, without any criticisms related to button functionality.
    - This essentially confirmed that the current button hover/activation setup is a suitable heuristic, and so leaving that unchanged is currently the best option.
* Overall, the reviews mostly consisted of recognising a good start. This tells me that the limited sections f the project I have developed so far work well and may not need much changing.

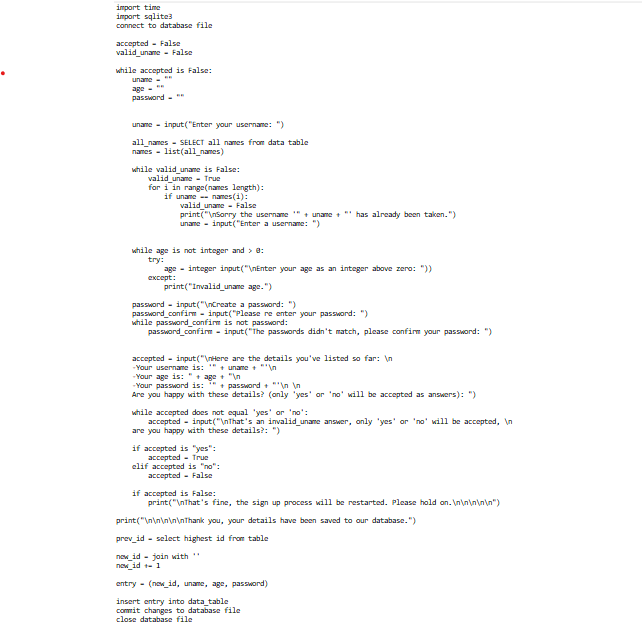
## Third prototype.

### Analysis.

* Using stakeholder feedback on the previous prototype, it was conclude that the functionality that has already been implemented should serve as satisfactory to my end user. Therefore, it’s now time to move on.
* Up until now, all the menus have been created and are running, with GUI button links and smooth menu navigation to hold the program together. The focus for this prototype is menu functionality. In other words, this prototype will introduce full functionality for all the menus I have created.
* Here’s what I plan to implement for each menu:
  + Main menu: As this menu, like the others, meets it’s intended navigation functionality, the only thing left is to implement a ‘start’ button that will signal the starting of the main game. The main is due to start development in the next prototype.
  + Main settings menu: For this menu I plan to implement display brightness and audio settings. With this, the user should be able to adjust GUI brightness, and the audio volume levels of the program. As audio hasn’t yet been implemented, being able to adjust a variable representing the audio level percentage (as integer from 0 to 100) will suffice for now.
    - Accessibility menu: For this menu I plan to implement a single switch that activates colour-blind mode. Colour-blind mode will be represented with a Boolean variable, actually adjusting RGB values to have this affect the GUI will be implemented later.
    - Game modifiers menu: For this menu I want to implement ‘double gravity’, ‘enemy frenzy’, and ‘speed demon’ switches. These game modifiers will change the value of Boolean variables representing said modifiers. This will in turn later be implemented to change how the main game plays (e.g. having a ‘speed\_demon\_mode’ variable set to ‘True’ will double player and enemy movement speed.)
  + Account menu: For this menu I plan to implement a fully functioning account system with it’s own database in a separate file. The ‘sign in’ and ‘sign up’ menus would help with this by allowing the user enter details to help these functions get carried out. I would also need to implement a ‘log out’ button which signs the user out of the account. Each of these account menus and sub-menus would also need a confirmation button so that the program knows to accept the entered details.

### Design.

Here’s some pseudocode for the signing up system:



* I plan to implement this into pygame, in the ‘sign up’ menu module. To do this I will use a combination of python and SQL.

### Test data.

|  |  |  |  |
| --- | --- | --- | --- |
| Text inputs boxes: | | | |
| Test data type: | Test data: | Expected output: | Justification: |
| Valid | Press mouse outside box | Box deactivates | Test whether text box can be deactivated as intended. |
| Valid | Press mouse inside box coordinates | Box activates | Test whether text box can be activated as intended. |

|  |  |  |  |
| --- | --- | --- | --- |
| Sign up menu - password | | | |
| Test data type: | Test data: | Expected output: | Justification: |
| Boundary | ‘xcZ\76nope’  ‘Xc8\*’ | The system should accept this as a valid password, no errors should be raised even though the password is at the character limit boundarty. | This is a test to ensure that valid passwords are accepted. |
| Invalid | ‘124djdkfmS’  ‘124djdkfm&’  ‘ASDdjdkfm&’ | The system should recognise that either an uppercase letter, lowercase letter, symbol, or number hasn’t been used, or the password is outside of the character limit and count the password as invalid. | This is a test to make sure that the password validation works as intended. |

The password has the most validation and so it’s the account detail that is most important to test, as the process is more complex and therefore less robust.

### Implementation (leave until coding is finished).

* Used text input library made by Nearoo [7] on Github for account text boxes.
* Used art made by Alucard [8] on opengameart for background art.
* Used music made by FoxSynergy [9] on opengameart for GUI BGM.
* Used sfx made by Fupi [10] on opengameart for button press SFX.
* Used stack overflow syntax from the4thiceman [11] to play multiple sounds at the same time.

### Testing (empty).

### Remedial actions (take error/solution screenshots whilst coding).

### Evaluation (empty).

#### Stakeholder contact (empty).

# References

|  |  |
| --- | --- |
| [1] | Haz. [Online]. Available: https://stackoverflow.com/questions/16044229/how-to-get-keyboard-input-in-pygame. |
| [2] | B. Lewandowski. [Online]. Available: https://stackoverflow.com/questions/20842801/how-to-display-text-in-pygame. |
| [3] | “Program arcade games,” [Online]. Available: http://programarcadegames.com/index.php?chapter=introduction\_to\_graphics&lang=en. |
| [4] | sloth, “Stack overflow,” [Online]. Available: https://stackoverflow.com/questions/10990137/pygame-mouse-clicking-detection. |
| [5] | dmitri13, “www.flaticon.com,” [Online]. Available: https://www.flaticon.com/authors/dmitri13. |
| [6] | D. Gandy, “Flaticon,” [Online]. Available: https://www.flaticon.com/authors/dave-gandy. |
| [7] | Nearoo, “Github,” [Online]. Available: https://github.com/Nearoo/pygame-text-input. |
| [8] | Alucard, “opengameart,” [Online]. Available: https://opengameart.org/content/city-background-repetitive-3. |
| [9] | FoxSynergy, “opengameart,” [Online]. Available: https://opengameart.org/content/blue-space. |
| [10] | Fupi, “opengameart,” [Online]. Available: https://opengameart.org/content/8bit-menu-select. |
| [11] | The4thIceman, “Stack overflow,” [Online]. Available: https://stackoverflow.com/questions/42393916/how-can-i-play-multiple-sounds-at-the-same-time-in-pygame. |